What do Clones do on Mother's Day? An Overview of the Strange New World of the Clone Table

Rich Wolfson

Principal Consultant

September 15, 2010

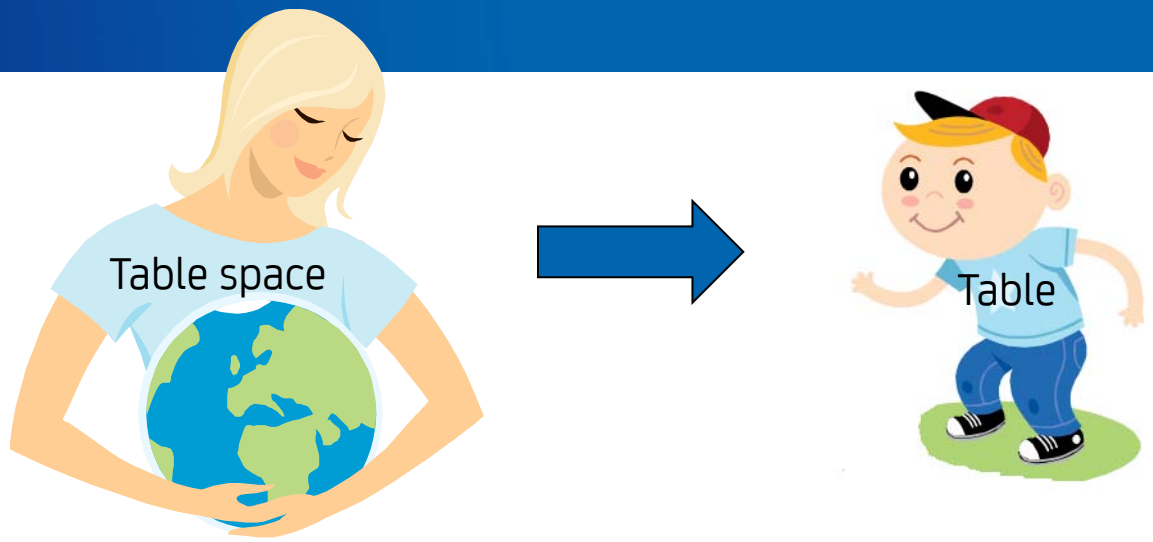
Abstract

— In previous releases of DB2, users expressed the need for functionality similar to an online LOAD REPLACE utility. Introduced in DB2 9, clone table support provides users with the ability to generate a table with the *exact same* attributes as a table that already exists in the current DB2 subsystem. The clone is created within the same universal table space as the base table, and is structurally identical to the base table in every way. Once created, a clone table can be manipulated independently of the base table, and you can exchange data between the base table and the clone table. Clone tables thus provide the functional equivalent of an online LOAD REPLACE utility, allowing users to, for example, reload table data multiple times a day while still being able to access the existing data with read-only operations. This presentation will examine the new clone table feature, discussing concepts, features, commands and restrictions (including the new UTS table space type).

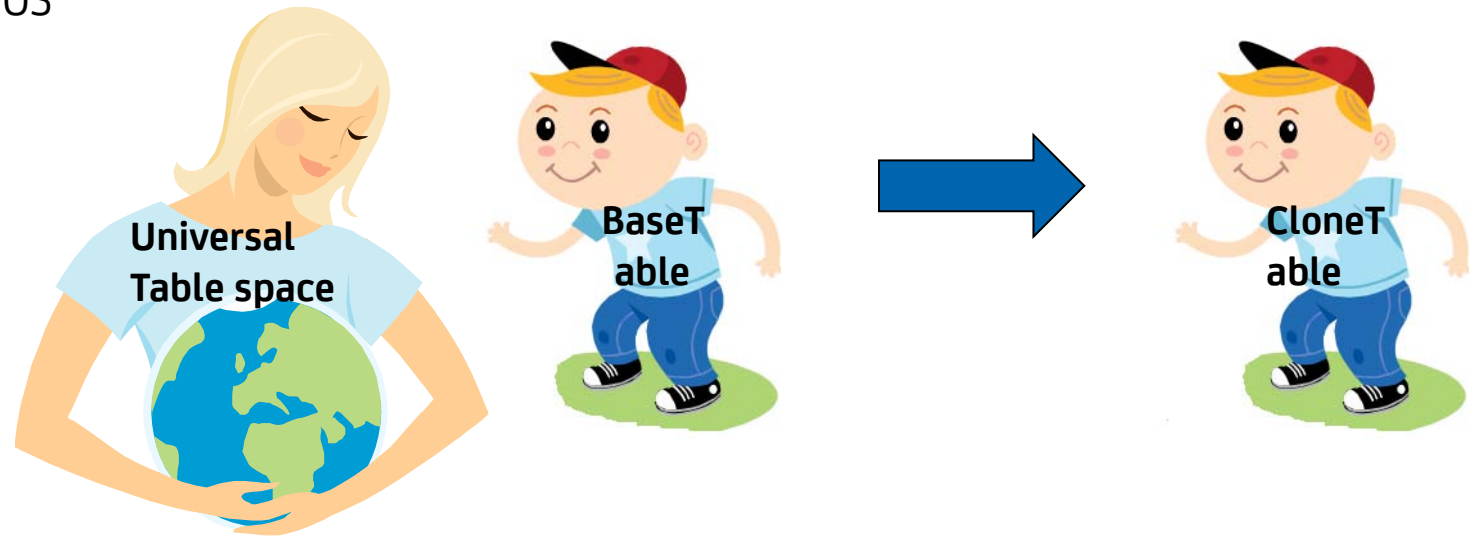
Agenda

- What is a clone table and why would I want one?
- Clone table concepts and overview
- First - an overview of the Universal Table Space
- Creating a clone table
- Commands and clone tables
- Exchanging clone and base table
- Dropping a clone table
- Restrictions
- Backup and recovery involving clone tables
- Utilities and clone tables
- Summary

Pre-DB2 9 z/OS



DB2 9 z/OS



What is a clone table and why would I want one?

- ***Clone*** - In horticulture and biology, any organism whose genetic information is identical to that of a "mother organism" from which it was created.

- In DB2, in a nutshell, “clone” tables provide faster replacement of one table with another
- A clone table is a table with the EXACT SAME attributes, structure, data as an already existing table
  - Created in the same table space as the base table
    - Base table must be created in new DB2 9 “Universal Table Space” (UTS)

What is a clone table and why would I want one? (cont'd)

- Clones can be worked with independently
  - You can insert or load data into the clone table
  - You can exchange the clone table name with the current table name

— Clone tables provide the functional equivalent of an online LOAD REPLACE utility
  - No outage / data is still accessible
  - Clones allow users to, for example, reload table data multiple times a day while still being able to access the existing data with read-only operations

Clone table concepts and overview

- Clones are structurally identical to the base table in every way
  - Same number of columns, column names, data types, check constraints, etc.
  - Created with same indexes, before triggers, LOB objects, etc.
    - DB2 automatically creates the clone objects when it creates the clone table
- Clones can have different name/schema than their base table, but are referred to by the same names as those that are used for base table objects

First, an overview of the Universal Table Space (UTS)

- New table space type introduced in DB2 9
  - A follow-on to DB2 V8 partitioning changes
- Universal Table Spaces (UTS) are both segmented and partitioned, and combine the advantages of both
  - Yes, ***both***
- Two types of UTS:
  - Partitioned-by-growth (PBG)
  - Range-partitioned (PBR)

Universal Table Space: Partition-by-Growth - PBG

- Partitioned by *data* growth
- Useful for table spaces that are likely to exceed the 64 GB limit for simple or segmented table spaces
  - But do not have a suitable partitioning key
  - Can grow up to 128 TB
- Segmented tables can be partitioned as they grow, without the need for key ranges

Universal Table Space: Partition-by-Growth - PBG (cont'd)

- Partition-by-growth table spaces begin life as a single-partition table space and automatically grow partitions as needed to accommodate data growth
  - When first partition fills, next partition is created, etc.
  - Continues up to max # of partitions defined when created
- Data structure is similar to structure of a segmented table space
  - Better space management and mass delete capabilities than simple table spaces

Universal Table Space: Partition-by-Growth - PBG (cont'd)

- Partition-by-growth table spaces can be created either explicitly or implicitly:
  - **Explicitly** – new keyword MAXPARTITIONS for CREATE TABLESPACE and ALTER TABLESPACE statements
  - **Implicitly** – PARTITION BY SIZE clause on CREATE TABLE statement
  - For partition-by-growth table spaces, column TYPE in SYSIBM.SYSTABLESPACE contains a 'G'

Universal Table Space: Partition-by-Growth - PBG (cont'd)

- Considerations and restrictions
  - Partition-by-growth table spaces must be storage group controlled
  - You cannot use the following options for partition-by-growth table spaces:
    - ALTER TABLE ADD PARTITION
    - ALTER TABLE ROTATE PARTITION
    - ALTER TABLE ALTER PARTITION
- A partition-by-growth table space can only contain one table
- Only NPIs are allowed on tables residing in partition-by-growth table spaces

Universal Table Space: Range-partitioned - PBR

- Two main points:
  - Based on partitioning ranges
  - Can contain only a single table
    - As is the case with regular partitioned table spaces
- Does not replace the existing (DB2 V8) “regular” partitioned table space!
- Supports same operations as regular partitioned or segmented table spaces
- Created by specifying **both** NUMPARTS and SEGSIZE in one CREATE TABLESPACE statement

Universal Table Space: Range-partitioned - PBR (cont'd)

- Some Considerations
  - All range-partitioned universal table spaces are LARGE table spaces
  - For range-partitioned table spaces, column TYPE in SYSIBM.SYSTABLESPACE contains an 'R'
  - You need to use the table-controlled partitioning syntax introduced with V8 to create range-partitioned universal table spaces

Create UTS

— Universal Tablespace is mandatory in order to Clone

```
CREATE  TABLESPACE TS1
 IN DB01
 USING STOGROUP SYSDEFLT
 ERASE NO FREEPAGE 0 PCTFREE 5
 BUFFERPOOL BP1
 LOCKSIZE ANY
 SEGSIZE 64 MAXPARTITIONS 4
 LOCKMAX SYSTEM
 CCSID EBCDIC ;

DSNT400I SQLCODE = 000
```

```
CREATE  TABLESPACE TS2
 IN DB01
 USING STOGROUP SYSDEFLT
 ERASE NO FREEPAGE 0 PCTFREE 5
 BUFFERPOOL BP1
 LOCKSIZE ANY
 SEGSIZE 64 NUMPARTS 4
 LOCKMAX SYSTEM
 CCSID EBCDIC ;

DSNT400I SQLCODE = 000
```

Partition by GROWTH

```
D91A.DSNDBD.DB01.TS1.I0001.A001

D91A.DSNDBD.DB01.TS2.I0001.A001
D91A.DSNDBD.DB01.TS2.I0001.A002
D91A.DSNDBD.DB01.TS2.I0001.A003
D91A.DSNDBD.DB01.TS2.I0001.A004
```

Partition by RANGE

ca technologies

Create Base Tables

— The base tables are created and a user-id granted

```
CREATE  TABLE PARGR04.IDUGTB1
(COL01 CHAR ( 4 ) NOT NULL
                  WITH DEFAULT
,COL02 INTEGER    NOT NULL
                  WITH DEFAULT
)
IN DB01.TS1
DATA CAPTURE CHANGES   ;
```

Partition by GROWTH
(use **PARTITION BY SIZE** on create table to describe when new part added

Partition by RANGE

```
CREATE  TABLE PARGR04.IDUGTB2
(COL01 CHAR ( 4 )    NOT NULL
                     WITH DEFAULT
                     FOR SBCS DATA
,COL02 INTEGER       NOT NULL
                     WITH DEFAULT
,COL03 CLOB ( 20 K ) NOT NULL
                     WITH DEFAULT
,COL04 ROWID
       GENERATED ALWAYS NOT NULL  )
partition by range  (col01 nulls last asc)
(partition    1  ending at  ('CCCC'),
 partition    2  ending at  ('KKKK'),
 partition    3  ending at  ('SSSS'),
 partition    4  ending at  ('ZZZZ') )
IN DB01.TS2
CCSID  EBCDIC  ;
```

```
GRANT SELECT     ON TABLE PARGR04.IDUGTB1    TO GREG ;
GRANT ALL        ON TABLE PARGR04.IDUGTB2    TO GREG ;
```

Creating a clone table

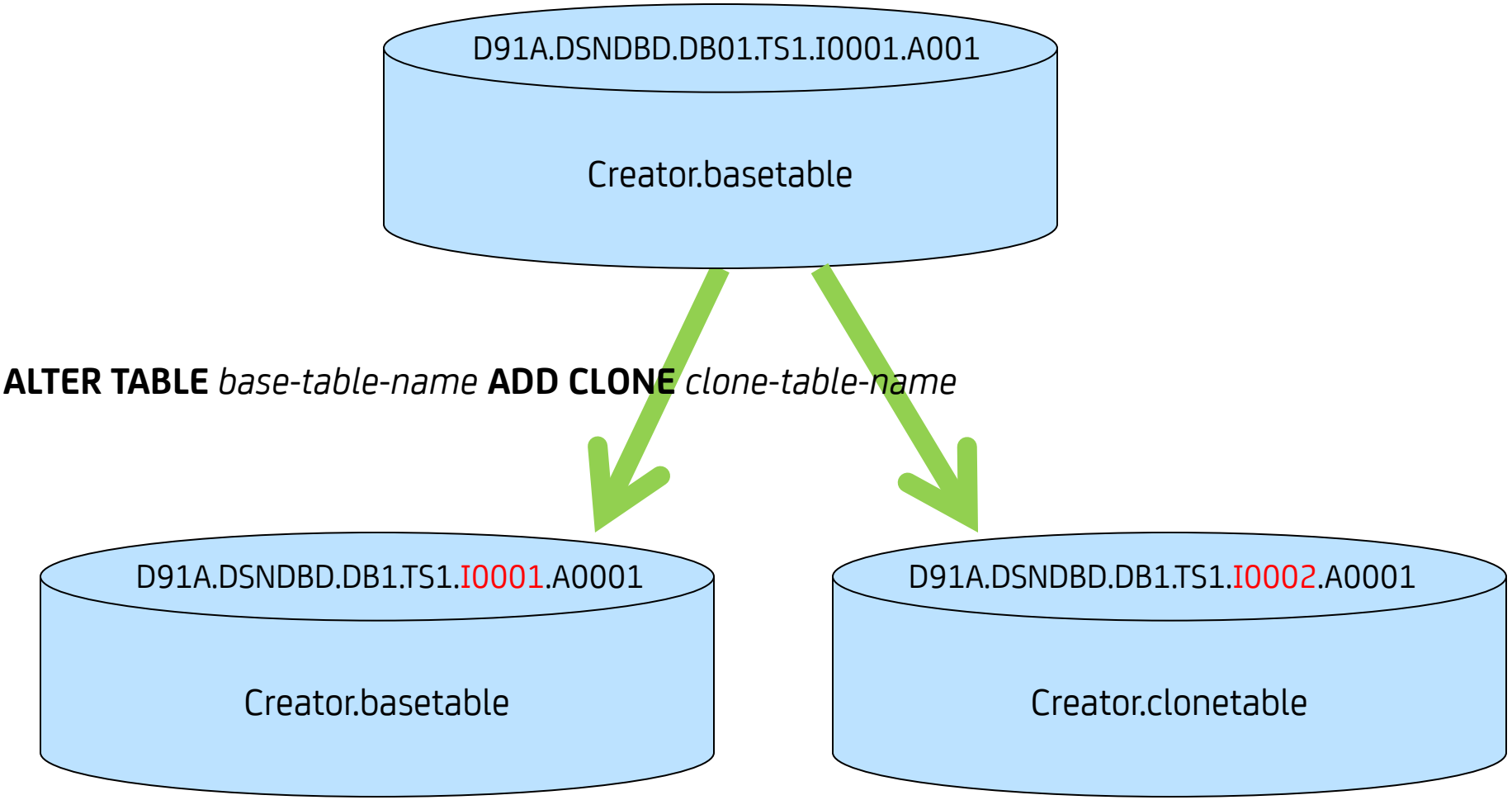
- *Remember,* can only clone base tables that have been created in UTSs
  - Partitioned-by-growth or range-partitioned
- *Remember,* additional clone objects will also be created - Indexes, constraints, triggers, etc.

- Create a clone table on an existing base table using new DB2 9 statement

ALTER TABLE *base-table-name* **ADD CLONE** *clone-table-name*

- The creation or drop of a clone table does not impact applications accessing base table data

Creating a clone table (cont'd)



Creating a clone table (cont'd)

- When a clone table is created, the OBID and the PSID is the same for both instances of the table space
- Clone table name and schema name can be chosen independently
- A clone table uses the statistics from the base table
  - RUNSTATS does not collect statistics on a clone table
  - Real-time Statistics (RTS) collects stats for both base and clone tables
- Indexes cannot be manually created on the clone table

Creating a clone table (cont'd)

- Example: GREGUTS1 created as a range-partitioned TS with table GREG9TAB and index GREG9IDX
- To create clone,

```
ALTER TABLE GREG.GREG9TAB ADD CLONE CLONEWAR;
```

- SELECT * FROM SYSIBM.SYSTABLES WHERE CREATOR = 'GREG';

```
  2 ROWS RETRIEVED
NAME                CREATOR             TYPE DBNAME
GREG9TAB            GREG                T    GSN8D91A
CLONEWAR            GREG                C    GSN8D91A
******************BOTTOM OF DATA**********************
```

- Note new Type 'C' for Clone

Commands and clone tables

- Commands that have been changed in lieu of clone objects
  - -DISPLAY DATABASE
  - -START DATABASE
  - -STOP DATABASE

- -DISPLAY DATABASE has been extended to indicate information for both base table objects and their clones
  - Base table objects with cloning are noted with an additional TYPE column character of 'B'
  - Cloned objects with a TYPE column character of 'C'...

Commands and clone tables (cont'd)

- -START DATABASE and -STOP DATABASE are extended in DB2 9 with the CLONE keyword so that you can specify clone objects to be started or stopped
- If CLONE keyword omitted, base table objects are started or stopped and the clone table objects will not be affected.
  - Note you can't START or STOP both base table and clone objects with just one command, you must issue two commands, one with and one without the CLONE keyword

Exchanging clone and base table

— To swap base table with clone table, use the new DB2 9 SQL statement:

EXCHANGE DATA BETWEEN TABLE *table-name1* **AND** *table-name2*

— Order of tables in the statement doesn't matter

- No data movement takes place when tables are exchanged
- Only instance numbers that point to base or clone table change
  - Pre-exchange clone table data instance number becomes the base table data instance number, and vice versa

— Note: EXCHANGE invalidates RTS

Exchanging clone and base table (cont'd)

- For example, a base table exists with the data set name *I0001.*:
  - The table is cloned and the clone's data set is initially named *.I0002.*
  - After an exchange, the base objects are named *.I0002.* and the clones are named *I0001.*
  - Each time an exchange happens, the instance numbers that represent the base and the clone objects change…
    - which immediately and effectively changes the data contained in the base and clone tables and indexes

Exchanging clone and base table (cont'd)

— Using our previous example:

-DISPLAY DATABASE(GSN8D91A) SPACENAM(*)

```
NAME     TYPE PART   STATUS              PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
GREGUTS1 TSB1  0001 RW
    -THRU      0044
GREGUTS1 TSC2  0001 RW
    -THRU      0044
```

SELECT * FROM SYSIBM.SYSTABLES WHERE TSNAME = 'GREGUTS1';

```
2 ROWS RETRIEVED
  NAME                CREATOR             TYPE DBNAME
  GREG9TAB            GREG                T    GSN8D91A
  CLONEWAR            GREG                C    GSN8D91A
```

ca technologies

Exchanging clone and base table (cont'd)

EXCHANGE DATA BETWEEN TABLE GREG9TAB AND CLONEWAR

```
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

-DISPLAY DATABASE(GSN8D91A) SPACENAM(*)

```
NAME     TYPE PART  STATUS              PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ------------------ -------- -------- -------- -----
GREGUTS1 TSB2  0001 RW
    -THRU      0044
GREGUTS1 TSC1  0001 RW
    -THRU      0044
```

After the exchange, the clone table can be read while the base table is free to do its own thing!

users

ca technologies

Dropping a clone table

- You can drop a clone table at any time
  - After or before an EXCHANGE
- Cannot use DROP TABLE statement, but rather:

ALTER TABLE *base-table-name* **DROP CLONE**

- This will drop the clone table, its indexes, LOB structures, and delete the underlying VSAM data sets from the current server
  - Base table and its objects remain unchanged
- If a base table that has clone objects is dropped, the associated clone objects are also dropped
  - DROP TABLESPACE will drop the whole shooting match (so, of course, will DROP DATABASE)

Restrictions

- To create a clone table, the *base* table must:
  - Reside in a UTS (partition-by-growth or range-partitioned table space)
  - Reside in a DB2-managed table space (i.e., STOGROUP-controlled
  - Be the only table in the table space
  - Not already have a clone table defined to it
  - Not have any referential constraints
  - Not have any AFTER triggers defined
  - Not be a materialized query table (MQT)

Restrictions (cont'd)

- To create a clone table, the *base* table must ALSO:
  - Not be a created global temporary table or declared global temporary table
  - Not have any data sets that are not yet created
    - IE, can't be base tables that reside in table spaces created using DEFINE NO – thus not yet using VSAM
  - Not have an incomplete definition
  - Not have any pending alterations or active versioning
    - OLDEST_VERSION and CURRENT_VERSION in SYSIBM.SYSTABLESPACE must be identical

Restrictions (cont'd)

- What cannot be cloned:
  - Views
  - Authorizations
  - Catalog and directory tables
  - Real-time Statistics (RTS) tables
- Before triggers cannot be created on the clone table
- Indexes cannot be created on a clone
- You cannot rename a base table that has a clone relationship
- You cannot alter any table or column attributes of a base table or clone table when the objects are involved with cloning
- The maximum number of partitions cannot be altered when a clone table resides in a partition-by-growth table space

RUNSTATS & EXPLAIN

- RUNSTATS can only be executed against BASE objects

```
RUNSTATS TABLESPACE GSN8D91A.GREGUTS1 INDEX(ALL)
```

- Not against the CLONE

```
RUNSTATS TABLESPACE GSN8D91A.GREGUTS1 TABLE(GREG.CLONEWAR)
COLUMN(ALL)

RUNSTATS UTILITY MAY NOT BE RUN AGAINST CLONE OBJECT
GREG.CLONEWAR
UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

- If “refresh” of data results in very “different data” (influence Access Path selection) – a subsequent RUNSTATS after EXCHANGE might be needed
- EXPLAINs can be executed against both base and clone

Backup and recovery involving clone tables

- When performing recovery on a clone table that has been exchanged, you can use an image copy that was made prior to an exchange
  - Note, no point in time recovery is possible prior to the most recent exchange
- Clone tables spaces and index spaces are stored in separate physical data sets
  - Can (must) be copied and recovered separately

Utilities and clone tables

- In DB2 9, many utilities have been extended to accommodate clone tables
- Specifying the CLONE keyword in the following utilities:
  - **COPY**
    - Indicates to copy only the clone table data in specified table spaces or index spaces
    - SYSCOPY INSTANCE number illustrates which VSAM dataset has been copied
  - **COPYTOCOPY**
    - Asks DB2 just to process image copy data sets that were taken against clone tables or indexes on clone tables

Utilities and clone tables (cont'd)

- Specifying the CLONE keyword in the following utilities:
  - Online **CHECK DATA**
    - Indicates that CHECK DATA is to check constraints for inconsistencies between the clone table data and the corresponding LOB data
  - **CHECK INDEX**
    - DB2 will check only the specified indexes that are on clone tables
  - **CHECK LOB**
    - Indicates that you want DB2 to check the LOB table space data for only the clone table, not the LOB data for the base table

Utilities and clone tables (cont'd)

- Specifying the CLONE keyword in the following utilities:
  - **DIAGNOSE**
    - Tells DB2 to display information for only the specified objects that are table clones, table spaces that contain tables clones, indexes on table clones, or index spaces that contain indexes on table clones
  - **MERGECOPY**
    - Only those image copy data sets that were taken against clone objects are processed
  - **MODIFY RECOVERY**
    - DB2 will delete all corresponding SYSCOPY and SYSLGRNX entries for your clone objects

Utilities and clone tables (cont'd)

- Specifying the CLONE keyword in the following utilities:
  - **LISTDEF**
    - Indicates that the INCLUDE or EXCLUDE expression will only return clone tables, table spaces that contain clone tables, indexes on clone tables, or index spaces that contain indexes on clone tables.
  - **QUIESCE**
    - Indicates that you want to create a quiesce point for only the specified table spaces that contain clone tables

Utilities and clone tables (cont'd)

- Specifying the CLONE keyword in the following utilities:
  - **REBUILD INDEX**
    - Only indexes on clone tables are rebuilt
    - If CLONE omitted, only base table index is rebuilt
    - The keywords CLONE and STATISTICS are mutually exclusive
      - DB2 does not collect statistics for table clones
  - **RECOVER**
    - DB2 recovers only clone table data in the specified table space, or the specified index spaces that contain indexes on clone tables
    - It is not possible to recover to a point prior to EXCHANGE
    - Taking an image copy after EXCHANGE might be a good idea

Utilities and clone tables (cont'd)

- Specifying the CLONE keyword in the following utilities:
  - **REORG INDEX**
    - Asks DB2 to reorganize only the specified index spaces that contain indexes on clone tables
  - **REORG TABLESPACE**
    - DB2 will only reorganize table clones from the specified table spaces
    - Since you can't collect statistics on clone tables, specifying both CLONE and STATISTICS is not allowed

Utilities and clone tables (cont'd)

- Specifying the CLONE keyword in the following utilities:
  - **REPAIR**
    - Indicates to REPAIR only those objects that are: table spaces that contain clone tables, indexes on clone tables, or index spaces that contain indexes on clone tables
    - You can't specify CLONE on REPAIR VERSIONS, because table clones do not have versions
  - **REPORT**
    - Will return information for only the specified objects that are table spaces that contain clone tables, indexes on clone tables, or index spaces that contain indexes on clone tables

Utilities and clone tables (cont'd)

- **UNLOAD**
  - If you specify the name of the clone table in the FROM TABLE clause, you don't need to specify the CLONE keyword to unload the clone table
  - If you specify the table space name in your UNLOAD syntax, you must specify CLONE to unload data from the clone table
- **DSN1COPY**
  - Specify the dataset instance name
- **DSN1LOGP**
  - No CLONE keyword; instead, uses PSID high order bit values in the DB2 log to determine which data manipulation belongs to the base table of a table space and which one belongs to the clone table

Summary

- A clone table is a table with the exact same attributes as an existing table… structurally identical
- Clone tables are created within the same Universal Table Space as the base table
- Two types of Universal Table Space: Partition-by-growth and range-partitioned
- Clone tables can be manipulated independently of their corresponding base tables
- Clone and base tables can be exchanged: No data movement takes place; instance numbers are swapped
- Clone tables provide the functional equivalent of an online LOAD REPLACE

Thank you

richard.wolfson@ca.com

ca technologies